AMENDMENTS TO THE CLAIMS

Kindly replace the claims as follows.

1. (previously amended) A microprocessor and support software, comprising:

an instruction pipeline designed to execute instructions of an instruction set, control-transfer instructions of the instructions being instructions defined to transfer execution control of a computer from a source instruction to a destination instruction, control-flow instructions of the instruction set being classified into a relatively small plurality of classes relative to the number of instruction opcodes executable by the instruction pipeline, most divisions in the classification being based on a static encoding of control-flow instructions executed, with at most minor divisions in the classification being based on dynamic or data-dependent execution behavior;

a storage register designed to store, and updating circuitry active during execution of a program on the microprocessor, designed to record into the storage register, as part of the execution of control-flow instructions of the instruction set and without software intervention, a value reflecting the class, from among the encoding-based classification, of a control-flow instruction recently executed by the pipeline;

software programmed to adjust the storage contents of the computer to reestablish in the context of the destination instruction a context logically equivalent to the context of the computer at the time of the control transfer instruction, the adjustment being determined, at least in part, by a classification of the control-transfer instruction; and

invoking circuitry to invoke the software before executing the destination instruction of at least some of the control transfer instructions.

2. (original) The microprocessor of claim 1, wherein:

at least three-quarters of the classification divisions are defined solely by static encoding of instructions executed;

at least some of the classification divisions are defined by the opcode of the instructions; and

at least some of the classification divisions are defined by immediate values of instructions.

3. (original) The method of claim 1, wherein:

at least four of the classification divisions distinguish different classes of jump instructions emitted by a typical compiler for the computer primarily to effect intra-subprogram transfers; and

at least half of the classification divisions distinguish classes of control transfers that either (a) induce a context switch, (b) are emitted by a typical compiler for the computer primarily to effect a control transfer between subprograms, or (c) effect argument passing or return as a side effect of the control transfer.

4. (original) The method of claim 1, wherein:

at least some of the immediate values defining the classification divisions are branch displacements; and

at least some of the immediate values defining the classification divisions are immediate values having no effect on the execution of the instructions in which they are encoded, except to update the storage register.

5. (original) The method of claim 1, wherein at least some instructions of the computer are assigned to a classification division ignored by the circuitry, execution of instructions in this ignored classification leaving intact the previous contents of the storage register.

6. (original) A method, comprising:

classifying control-flow instructions of a computer instruction set into a plurality of classes; and

during execution of a program on a computer, as part of the execution of instructions of the instruction set, updating a record of the class of the classified control-flow instruction most recently executed.

7. (original) The method of claim 6, further comprising:

detecting when the computer's execution flows from code observing a first data storage convention to code observing a second data storage convention, the record recording a class of instruction effecting the transitional execution flow; and

in response to the detecting, altering the data storage content of the computer to create a program context under the second data storage convention that is logically equivalent to a pre-alteration program context under the first data storage convention, the altering process being determined, at least in part, by the instruction classification record.

8. (original) The method of claim 7, wherein:

one of the two data storage conventions is a register-based calling convention, and the other data storage convention is a memory stack-based calling convention.

9. (original) The method of claim 8, wherein:

at least four of the classification divisions distinguish different classes of jump instructions emitted by a typical compiler for the computer primarily to effect intra-subprogram transfers; and

at least half of the classification divisions distinguish classes of control transfers that either (a) induce a context switch, (b) are emitted by a typical compiler for the computer primarily to effect a control transfer between subprograms, or (c) effect argument passing or return as a side effect of the control transfer.

10. (original) The method of claim 8, wherein:

in some of the control-flow instructions, the classification is dynamically determined based on a full/empty status of a register.

11. (original) The method of claim 6, wherein:

in some of the control-flow instructions, the classification is encoded as an immediate value of instructions, the immediate value having no effect on the execution of the instruction in which it is encoded, except to update the class record; and

in some of the control-flow instructions, the classification is statically determined by the opcode of the instructions.

12. (original) The method of claim 6, wherein:

at least four of the classification divisions distinguish different classes of jump instructions emitted by a typical compiler for the computer primarily to effect intra-subprogram transfers; and

at least half of the classification divisions distinguish classes of control transfers that either (a) induce a context switch, (b) are emitted by a typical compiler for the computer primarily to effect a control transfer between subprograms, or (c) effect argument passing or return as a side effect of the control transfer.

13. (original) A method, comprising:

executing a control-transfer instruction that transfers execution control of a computer from a first execution context to a destination instruction for execution in a second execution context;

before executing the destination instruction, reconfiguring the storage state of the computer to reestablish under the second execution context the logical state of the computer as interpreted under the first execution context;

the reconfiguring being determined, at least in part, by a classification of the control-transfer instruction.

14. (original) The method of claim 13, wherein the destination instruction is an entry point to an off-the-shelf binary for an operating system coded in an instruction set non-native to

the computer, the non-native instruction set providing access to a reduced subset of the resources of the computer.

15. (original) The method of claim 13, wherein:

at least four of the classification divisions distinguish different classes of jump instructions emitted by a typical compiler for the computer primarily to effect intra-subprogram transfers; and

at least half of the classification divisions distinguish classes of control transfers that either (a) induce a context switch, (b) are emitted by a typical compiler for the computer primarily to effect a control transfer between subprograms, or (c) effect argument passing or return as a side effect of the control transfer.

16. (original) The method of claim 13, wherein:

in some of the control-flow instructions, the classification is encoded as an immediate value of instructions, the immediate value having no effect on the execution of the instruction in which it is encoded, except to update the class record.

17. (original) The method of claim 13, wherein:

in some of the control-flow instructions, the classification is statically determined by the opcode of the instructions; and

in other of the control-flow instructions, the classification is dynamically determined with reference to a state of a processor register and/or data register of the computer.

18. (original) The method of claim 17, wherein:

for some of the control-flow instructions, the dynamic classification is determined based on a full/empty status of a register.

19. (original) The method of claim 13:

wherein one of the two data storage conventions is a register-based calling convention, and the other data storage convention is a memory stack-based calling convention;

and further comprising:

detecting when the computer's execution flows from code observing the register-based calling convention to code observing the memory stack-based calling convention, the record recording the classification of the instruction effecting the transitional execution flow; and

in response to the detecting, altering the data storage content of the computer to create a program context under the memory stack-based calling convention that is logically equivalent to a pre-alteration program context under the register-based calling convention [convention], the altering process being determined, at least in part, by the instruction classification record.

20. (original) The method of claim 19, wherein:

at least four of the classification divisions distinguish different classes of jump instructions emitted by a typical compiler for the computer primarily to effect intra-subprogram transfers; and

at least half of the classification divisions distinguish classes of control transfers that either (a) induce a context switch, (b) are emitted by a typical compiler for the computer primarily to effect a control transfer between subprograms, or (c) effect argument passing or return as a side effect of the control transfer.

21. (original) The method of claim 19, wherein:

in some of the control-flow instructions, the classification is encoded as an immediate value of the instructions, the immediate value having no effect on the execution of the instruction in which it is encoded, except to update the class record.

22. (original) The method of claim 19, wherein:

in some of the control-flow instructions, the classification is statically determined by the opcode of the instructions; and

in other of the control-flow instructions, the classification is dynamically determined based on a full/empty status of a register.

23. (original) The method of claim 19, wherein:

the rearranging is performed by an exception handler, the handler being selected by an exception vector based at least in part on the instruction classification record.

24. (previously amended) A microprocessor, comprising:

an instruction pipeline designed to execute instructions of an instruction set, the instructions of the instruction set being classified into a relatively small number of classes relative to the number of instruction opcodes executable by the instruction pipeline, most divisions in the classification being based on a static encoding of instructions executed, with at most minor divisions in the classification being based on dynamic or data-dependent execution behavior;

a storage register designed to store, and circuitry designed to record without software intervention into the storage register, a value reflecting the class, from among the encoding-based classification, of an instruction recently executed by the pipeline.

25. (original) The microprocessor of claim 24, wherein at least three-quarters of the classification divisions are defined solely by static encoding of instructions executed.

26. (original) The microprocessor of claim 25, wherein:

some of the classification divisions are defined by the opcode of the instructions; and

some of the classification divisions are defined by immediate values of instructions.

27. (original) The microprocessor of claim 26, wherein:

at least four of the classification divisions distinguish different classes of jump instructions emitted by a typical compiler for the computer primarily to effect intra-subprogram transfers; and

at least half of the classification divisions distinguish classes of control transfers that either (a) induce a context switch, (b) are emitted by a typical compiler for the computer primarily to effect a control transfer between subprograms, or (c) effect argument passing or return as a side effect of the control transfer.

28. (original) The microprocessor of claim 26, wherein at least some of the immediate values defining the classification divisions are branch displacements.

29. (original) The microprocessor of claim 26, wherein at least some of the immediate values defining the classification divisions are immediate values having no effect on the execution of the instructions in which they are encoded, except to update the storage register.

30. (original) The microprocessor of claim 24, wherein:

at least some of the classification divisions are determined by dynamic reference to a state of architecturally-visible processor registers and/or general purpose registers of the computer.

31. (original) The microprocessor of claim 30, wherein the state of the register is a state maintained distinct from a data content of the register.

32. (original) The microprocessor of claim 24, wherein at least some instructions of the computer are assigned to a classification division ignored by the circuitry, execution of instructions in this ignored classification leaving intact the previous contents of the storage register.

33. (original) The microprocessor of claim 24, wherein the number of classes is at most 32.

34. (original) The microprocessor of claim 24, further comprising hardware or software designed to use the content of the storage register to annotate a descriptor in a profile of the execution of a program running on the microprocessor.

35. (new) The method of claim 6, wherein the execution of the program on the computer is performed in the hardware instruction pipeline of the computer.